```
// https://s3.amazonaws.com/nrbook.com/AandS-a4-v1-2.pdf
// https://academicjournals.org/article/article1380546086_Ozdogan%20and%20Nalcaci.pdf#:~:text=wide%20range%20of%20quantum%20numbers,%20orbital%20exponents%20and%20internuclear%20distances.
// https://mathworld.wolfram.com/SphericalHarmonic.html

using System;

namespace nQuadrature
{
    // careful: we are using FORTRAN type arrays

    class Integrands
    {
        // simple integrands

        public double[] aa1 = { 0, 0, 0, 0, -1, -1, -1 };
        public double[] bb1 = { 0, 2, 1, 0.5 * Math.PI, 1, 1, 1 };
        public double exact1 = 1.434761888397263;

        public double F1(double[] x)
        {
            return x[1] * x[2] * x[2] * Math.Sin(x[3]) /
                (4 + x[4] + x[5] + x[6]);
        }

        public double[] aa2 = { 0, 0, 0, 0, 0 };
        public double[] bb2 = { 0, 1, 1, 1, 2 };
        public double exact2 = 0.5753641449035616;

        public double F2(double[] x)
        {
            return x[3] * x[3] * x[4] * Math.Exp(x[3] * x[4]) *
                Math.Pow(x[1] + x[2] + 1, -2);
        }

        public double[] aa3 = { 0, 0, 0, 0 };
        public double[] bb3 = { 0, 1, 1, 1 };
        public double exact3 = 2.152142832595894;

        public double F3(double[] x)
        {
            return 8.0 / (1 + 2 * (x[1] + x[2] + x[3]));
        }

        // oscillating integrals

        public double[] aa4 = { 0, 0, 0, 0, 0, 0 };
        public double[] bb4 = { 0, Math.PI, Math.PI, Math.PI,
                                Math.PI, 0.5 * Math.PI };
        public double exact4 = 0.16E2;

```

```
51          public double F4(double[] x)
52          {
53              return Math.Cos(x[1] + x[2] + x[3] + x[4] + x[5]);
54          }
55
56          public double[] aa5 = { 0, 0, 0, 0, 0 };
57          public double[] bb5 = { 0, 1, 1, 1, 1 };
58          public double exact5 = 0.1839071529076452;
59
60          public double F5(double[] x)
61          {
62              return Math.Sin(10 * x[1]);
63          }
64
65          public double[] aa6 = { 0, 0, 0 };
66          public double[] bb6 = { 0, 3 * Math.PI, 3 * Math.PI };
67          public double exact6 = -0.4E1;
68
69          public double F6(double[] x)
70          {
71              return Math.Cos(x[1] + x[2]);
72          }
73
74          public double[] aa7 = { 0, 0, 0, 0 };
75          public double[] bb7 = { 0, 1, 1, 1 };
76          public double exact7 = 0.8630462173553432;
77
78          public double F7(double[] x)
79          {
80              return Math.Pow(x[1] + x[2] + x[3], -2);
81          }
82
83          public double[] aa8 = { 0, 0, 1, 2 };
84          public double[] bb8 = { 0, 1, 2, 3 };
85          public double exact8 = 15;
86
87          public double F8(double[] x)
88          {
89              return 8 * x[1] * x[2] * x[3];
90          }
91
92          public double[] aa9 = { 0, 2, -1, 1 };
93          public double[] bb9 = { 0, 3,  4, 0 };
94          public double exact9 = 755.0 / 4.0;
95
96          public double F9(double[] x)
97          {
98              return 4 * x[1] * x[1] * x[2] - x[3] * x[3] * x[3];
99          }
100
101         public double[] aaa = { 0, 0, 0, 0 };
102         public double[] bba = { 0, 3, 2, 1 };
```

```
103         public double exacta = 12;
104
105         public double Fa(double[] x)
106         {
107             return x[1] * x[2] + x[3];
108         }
109
110         public double Z = 2.0;
111         public double exactb = 5.0 * 2.0 / 8.0;
112         public double[] aab = { 0.0,
113             1.0e-3, 0.0, 0.0,
114             1.0e-4, 0.0, 0.0 };
115         public double[] bbb = { 0.0,
116             6.9379307e-1, Math.PI, 2.0 * Math.PI,
117             6.9379307e-1, Math.PI, 2.0 * Math.PI };
118
119         public double Fb(double[] x)
120         {
121             double x12 = x[1] - x[4];
122             double y12 = x[2] - x[5];
123             double z12 = x[3] - x[6];
124             double r12 = Math.Sqrt(
125                 x12 * x12 + y12 * y12 + z12 * z12);
126             double ps1 = Math.Pow(Z, 1.5) *
127                 Math.Exp(-Z * x[1]) / Math.Sqrt(Math.PI);
128             double ps2 = Math.Pow(Z, 1.5) *
129                 Math.Exp(-Z * x[4]) / Math.Sqrt(Math.PI);
130             double dV = x[1] * x[1] * x[4] * x[4] *
131                 Math.Sin(x[2]) * Math.Sin(x[5]);
132             return ps1 * ps2 * dV / r12;
133         }
134
135         public double R = 0.5, zeta = 5.8, zetap = 4.2;
136         public int n = 1, np = 1, l = 0, lp = 0, m = 0, mp = 0;
137         public double[] aac = { 0.0, 1.0e-8, 0.0, 0.0, 0.0, 1.0e-6, 0.0, 0.0 };
138         public double[] bbc = { 0.0, 3.4464444044e-1, Math.PI, 2.0 * Math.PI,
139             3.4464444044e-1, Math.PI, 2.0 * Math.PI };
140         public double theta = 30.0 * Math.PI / 180.0;
141         public double phi = 135.0 * Math.PI / 180.0;
142         public double exactc = 0.450897002422256;
143
144         public double Factorial(int n)
145         {
146             double F = 1.0;
147
148             for (int i = 2; i <= n; i++)
149             {
150                 F *= i;
151             }
152
153             return F;
154         }
```

```
155
156         public double N(double zeta, int n)
157         {
158             return Math.Pow(2.0 * zeta, n + 0.5) /
159                 Math.Sqrt(Factorial(2 * n));
160         }
161
162         public int KroneckerDelta(int m, int n)
163         {
164             return m == n ? 1 : 0;
165         }
166
167         public double PhiM(double phi, int m)
168         {
169             double c = 1.0 /
170                 Math.Sqrt(Math.PI + KroneckerDelta(m, 0));
171             double factor;
172
173             if (m >= 0)
174                 factor = Math.Cos(Math.Abs(m) * phi);
175             else
176                 factor = Math.Sin(Math.Abs(m) * phi);
177
178             return c * factor;
179         }
180
181         // real spherical harmonics for l = i, m = 0 where i = 0, 1, or 2
182         public double S(double theta, double phi, int l, int m)
183         {
184             if (l == 0 && m == 0)
185                 return 1.0 / Math.Sqrt(4.0 * Math.PI);
186             else if (l == 1 && m == 0)
187                 return Math.Sqrt(3.0 / (4.0 * Math.PI)) *
188                     Math.Cos(theta);
189             else if (l == 2 && m == 0)
190                 return Math.Sqrt(5.0 / (16.0 * Math.PI)) *
191                     (3.0 * Math.Pow(Math.Cos(theta), 2.0) - 1.0);
192             else
193                 return 0.0;
194         }
195
196         public double Chi(double zeta, double r,
197             double theta, double phi, int n, int l, int m)
198         {
199             return N(zeta, n) * Math.Pow(r, n - 1.0) *
200                 Math.Exp(-zeta * r) * S(theta, phi, l, m);
201         }
202
203         // two-center overlap integrals for Slater Type Orbitals (STOs)
204         public double Fc(double[] x)
205         {
206             double x12 = x[1] - x[4] + R;
```

```
207             double y12 = x[2] - x[5];
208             double z12 = x[3] - x[6];
209             double r12 = Math.Sqrt(
210                 x12 * x12 + y12 * y12 + z12 * z12);
211             double chi1 = Chi(zeta, x[1], theta, phi, n, l, m);
212             double chi2 = Chi(zetap, x[4] - R, theta, phi, np, lp, mp);
213             double dV = x[1] * x[1] * x[4] * x[4] *
214                 Math.Sin(x[2]) * Math.Sin(x[5]);
215             return chi1 * chi2 * dV / r12;
216         }
217     }
218 }
```